Санкт-Петербургский государственный университет

Математико механический факультет
Математическое обеспечение и администрирование информационных систем

Лучко Александр Юрьевич

Определение профессионального уровня программиста по github аккаунту

Дипломная работа

Научный руководитель:
к. ф.-м. н., доцент Графеева Н. Г.

Рецензент:
Горовой В.А к. т. н

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

The Faculty of Mathematics and Mechanics
Chair of Analytical Information Systems

Luchko Alexander

Analysis of programmers by github accounts

Graduation Thesis

Scientific supervisor:
associate professor Natalia Grafeeva

Reviewer:
candidate of engineering sciences Vladimir Gorovoy

Saint-Petersburg
2017

Оглавление

Введение

GitHub самая популярная система контроля версий для создания open source продуктов. Свою популярность данный сайт получил не только благодаря инструментам, обеспечивающим сам процесс разработки, но и благодаря встроенной системе социальных отношений [1]. Многие люди пользуются GitHub , чтобы создавать и поддерживать существующие продукты, но не меньшее количество людей приходит из-за популярности данного сервиса. Ввиду всего этого на GitHub происходят явления социального характера [1], которые не могут быть объяснены в терминах данной системы (например, пользователь с большим числом подписчиков, но совсем без программного кода).

Уже на протяжении нескольких лет GitHub активно развивается, в данный момент сложно найти разработчика, не исользующего данный сервис. Каждый программист, который зарегестрирован на GitHub, должен иметь профиль, содержащий важную информацию о нём. В связи с этим GitHub стал интенсивно использоваться HR менеджерами для поиска сотрудников.

Перечислим информацию, доступную в профиле программиста, котрую может использовать HR менеджер:

1)Organizations - организации в которых состоит пользователь;

2)Repositories - репозитории (репозиторий - "папка с кодом" какого-либо проекта);

3)Stars - количество звёзд, которые поставил пользователь другим репозиториям. Также можно просмотреть непосредственно эти репозитории;

4)Followers - количество тех, кто подписался на данного пользователя. Также можно просмотреть непосредственно этих пользователей;

5)Following - количество людей в подписке пользователя с возможностью просмотреть профили этих людей;

6)Popular repositories - самые популярные репозитории пользователя, которые были специально выбраны для экспозиции их на главной странице. В настройках пользователь может поменять список выбранных репозиториев (тогда их называют pinned repositories, то есть специально поставленные пользователем);

7)Contribution - поле, характеризующие вклад пользователя в какие-либо проекты за последний год. Является натуральным числом.

Ниже вы можете видеть, как это выглядит на сайте GitHub (элементы пронумерованы).

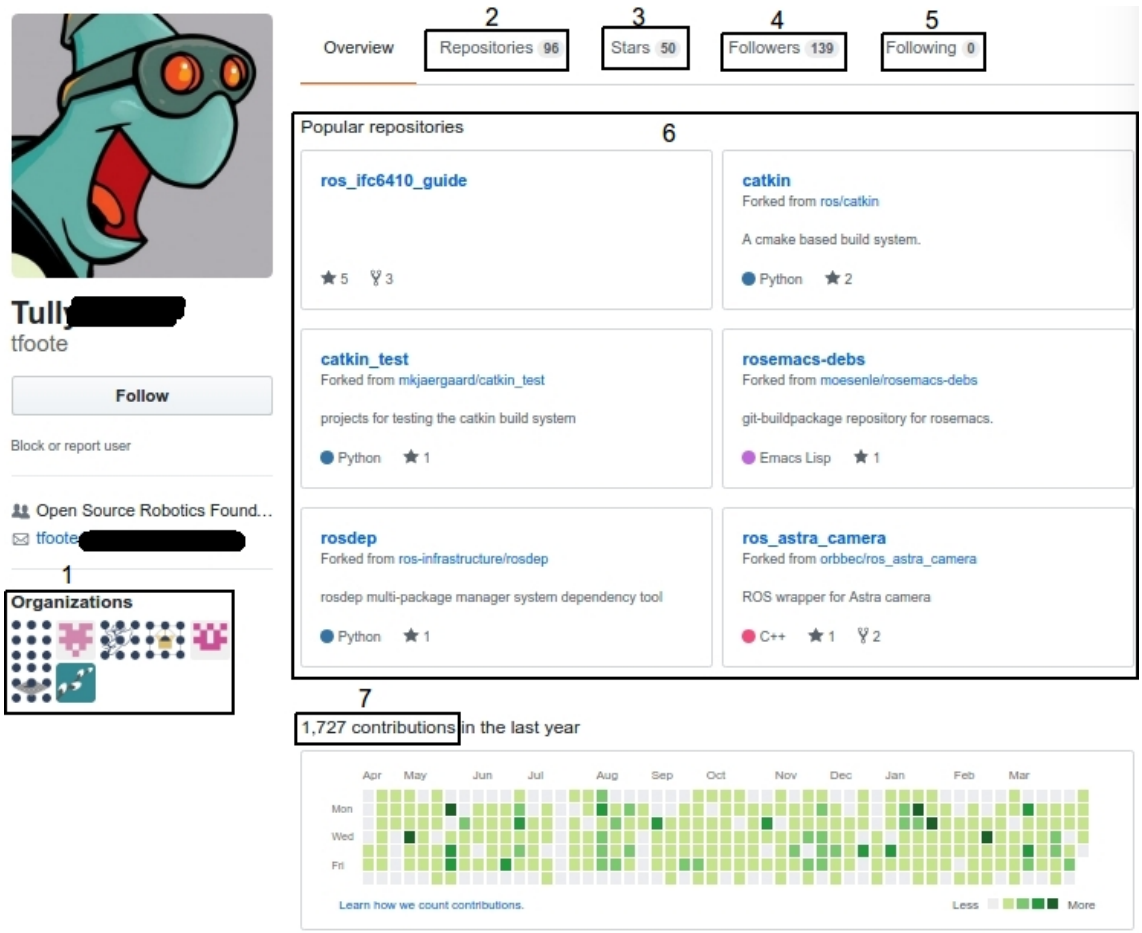


Рис. 1: Страница профиля программиста

Также можно просмотреть репозитории пользователя, которые содержат следующую информацию:

1)код проекта;

2)Watchers - количество людей, наблюдающих за репозиторием. Также можно просмотреть профили непосредственно этих людей;

3)Stargzers - количество пользователей поставивших звёздочку репозиторию с возможностью просмотреть принадлежащие им профили;

4)Forks - количество "копий" данного репозитория в терминах GitHub, с возможностью просмотреть данные копии.

Благодаря этой информации, менеджеры по персоналу могут в ручную оценивать любого пользователя. Обычно они смотрят количество подписчиков, которое говорит о популярности пользователя, в ручную просматривают код, пробегая по некоторым из его репозиториев, однако практически невозможно проводить полный анализ или сравнивать сразу нескольких кандидатов (ввиду объёма информации). В итоге данный анализ превращается в сравнение полей, доступных в профиле кандидата, что ненадёжно в смысле корректной оценки.

Очевидно, проблема невозможности HR менеджера пробежаться по всем данным

пользователя могла бы быть успешно решена при автоматизации данного процесса, но, к сожалению, сейчас не существует подобных подходов. Также, очевидно, что проблема автоматизации - это непростая задача, так как программисты могут проявлять одинаковую активность, иметь одно и тоже количество репозиториев, но писать существенно разный по ценности код.

Построение метрики, которая поможет в автоматизации оценки разработчиков, могло бы существенно упростить работу HR менеджеров, а также помочь выработать глубокое понимание того, как должна проводиться ручная оценка.

1. Существующие работы на данную тему

Как упоминалось ранее, в данный момент для GitHub не существует подходов автоматизированной оценки программистов. Единственное, что существует - это разного рода советы для HR менеджеров, некоторые подходы оценки программистов вручную.

В работе [3] предлагается сопоставление readme файлов репозиториев, куда делал вклад разработчик, и объявлений на работу с помощью выделения ключевых слов и их взвешивания. Это может быть полезно в поиске кандидатов на работу, но не помогает при оценивании умений программистов.

Также есть множество работ, объясняющих по каким критериям может проводится оценка и одобрение вклада программиста, в статье [7] показаны заметки по ручной оценке "newcomers" (новых разработчиков), даётся объяснение, по каким критериям стоит набирать программистов в open source проекты, а также какие критерии являются чаще всего употребительными для принятия их вкладов. Работа [7] является систематизированным обобщением многих статей на тему факторов, влияющих на создания и принятие вклада в репозиторий.

Существует также работа, показывающая, как измерить вклад программиста в конкретный репозиторий. То есть построена метрика оценки работы программиста в одном конкретном проекте [2].

В работе [4] используя технику анализа социальных ресурсов, предлагается измерять популярность репозитория, что в каком-то смысле будет похоже на технику, используемую в нашей работе.

Многие работы [1],[5],[6] фокусируют своё внимание на росте и долгосрочных перспективах развития github, исследуют тенденцию изменения характера социальных отношений, что даёт глубокое понимание внутренних закономерностей данного сервиса.

2. Постановка задач

В данной работе будут решены следующие задачи:

1)Построения метрики оценки пользователей по GitHub аккаунту;

2)Проведение анализа реальных пользователей Германии;

3)Выработка надёжного ручного алгоритма, который приближенно оценивает значения метрики по любому из пользователей.

Дальнейшее повествование будет соответствовать нумерации, приведённой в этой главе.

3. Метрика

В данной главе будет приведена метрика оценки пользователя (3.1), в которой будут активно использоваться характеристики, перечисленные во введении. В под главе (3.2) будут приведены конкретные примеры использования метрики. В конце главы будут указаны некоторые замечания о достоинствах и недостатках данной метрики (3.3).

3.1. Построение метрики

U(x) – будем обозначать метрику на множестве аккаунтов. То есть **Dom(U)** есть все аккаунты на GitHub.

Rr(y) – будем обозначать метрику на множестве репозиториев. То есть **Dom(Rr)** есть все репозитории на GitHub.

Оценка пользователя складывается из двух частей: оценка профиля и оценка всех вкладов. Формулой это можно изобразить так:

$$U(x) = log(profile\ estimation) + log(contribution\ estimation)$$

Разберём более подробно, что мы понимаем под оценкой всех вкладов (**contribution estimation**). Оценкой всех вкладов будет сумма каждого вклада (в терминах github вклад - это commit), умноженного на "вес" (популярность) репозитория, в который был сделан этот вклад. Метрика по репозиториям $Rr()$ как раз измеряет указанный "вес" репозитория (на формуле ниже $r.contribution$ - это все вклады пользователя в репозиторий r).

$$contribution\ estimation = \sum_{r\ \in\ x.repos()} Rr(r) * r.contribution$$

Тогда полная формула метрики пользователя может быть переписана так:

$$U(x) = log(profile\ estimation) + log(\sum_{r\ \in\ x.repos()} Rr(r) * r.contribution)$$

Где непосредственно репозиторий взвешивается с помощью сложения количества наблюдателей(watchers), количества людей, поставивших звезду репозиторию (stargazers) и числа копий (forks), сделанных по данному репозиторию, при этом каждая характеристика домножается на соотвествующий коэффицент.

$$Rr(r) = r.watchers * ratio\_watchers + (r.stargazers/1000) * r.forks$$

$$+r.stargazers * ratio\_stargazers$$

Теперь рассмотрим ($profile\ estimation$), который будет зависеть только от коли-

чества подписчиков с каким-то коэффицентом.

$$(profile\ estimation) = user.followers * ratio\_followers$$

Коэффиценты в формулах с приставкой ratio - это константы, которые должны быть подобраны. Всё это и есть полное определение метрики.

3.2. Примеры использования метрики

Покажем на практике, как ведёт себя метрика. Рассмотрим пользователей ALEXSSS, kinmanz, mbostock , tfoote, dhh, которых вы можете просмотреть непосредственно на сайте GitHub. Двое из них kinmanz и ALEXSSS использовали сервис очень редко: имеют всего два публичных репозитория, в которые делали вклады. В свою очередь mbstock, tfoote и dhh используют сервис активно, все они делают вклады в популярные репозитории и имеют немало подписчиков, особо среди них отличается mbstock, который делает огромное число вкладов в очень популярные репозитории и имеет больше всего подписчиков (порядка 14 тысяч). Ниже приведены оценки, полученные по метрике для данных пользователей:

ALEXSSS : 8.4
kinmanz : 21.3
mbostock : 185.5
dhh : 79.7
tfoote : 66.1

Как видим оценка коррелируется со словесным описанием. Мы можем наблюдать, что значение метрики у пользователя tfoote немного больше, чем у пользвателя с именем dhh. Важно заметить, что dhh имеет большее число подписчиков, чем tfoote, но по вкладам они примерно одинаковые, поэтому полученные оценки не сильно разнятся по отношению к друг другу.

3.3. Достоинства метрики

Метрика оказалась действительно эффективной в оценке пользователей GitHub. Преимущества данного подхода в том, что он фокусируется на реальных вкладах программиста, а не рассматривает только визуально доступные характеристики. В свою очередь можно решить, что слабое место метрики во взвешивании вкладов, так как эта часть вычислений может быть произведена по-другому: через инспекцию кода (анализ качества кода, употребления паттернов ООП и тд), однако, стоит заметить, что данный анализ намного сложнее и поэтому неприменим с практической точки

зрения. Сам подход, использующий основные характеристики репозитория, подразумевает, что они будут повышены, если репозитории активно развиваются, а значит и качество кода в данных репозиториях выше, что автоматически даёт необходимую инспекцию. Не меннее важное достоинство метрики - это возможность её реализации как рабочего инструмента, которым могут пользоваться HR менеджеры, то есть метрика выходит за пределы теоретической идеи и может приносить реальную пользу.

4. Проведения анализа пользователей Германии

Эта глава идёт сразу после определения метрики, так как анализ, проводимый здесь, будет использовать предыдущие результаты. В этой главе мы увидим, что составные части метрики, такие как оценка профиля (profile estimation) и оценка всех вкладов (contribution estimation) являются независимыми элементами, и для оценки пользователя не могут рассматриваться отдельно. Данное наблюдение подкрепит определение нашей метрики, а также даст понимание внутренней структуры данных пользователей Германии (4.2). Затем мы рассмотрим все доступные репозитории немецких пользователей и покажем зависимость между количеством у них звезд(stargazers), количеством подписчиков (watchers) и количеством копий(forks), что позволит нам сделать инетересное наблюдение о том (4.2), что данные величины одинаково показывают популярность репозитория.

4.1. Структура данных

Перед тем, как начать анализ, необходимо сказать пару слов о том, как были собраны данные, а также показать структуру этих данных.

Исходные данные для нашей системы были получены использованием GraphQL API, который недавно реализовал GitHub. Преимущество вышеупомянутого API в более экономичной и простой выборке данных графового строения. GraphQL подход претендует быть разумной альтернативой для REST подхода, который популярен на данный момент.

Структурно набор данных полностью копирует доступную информацию, указанную во введении, и является полями из профиля пользователя (количество подписчиков, организаций и тд), а также содержит характеристики обо всех репозиториях, доступных пользователю, с основной информацией о них (количество звёзд, наблюдателей, копий). Наши данные структурно выглядят так:

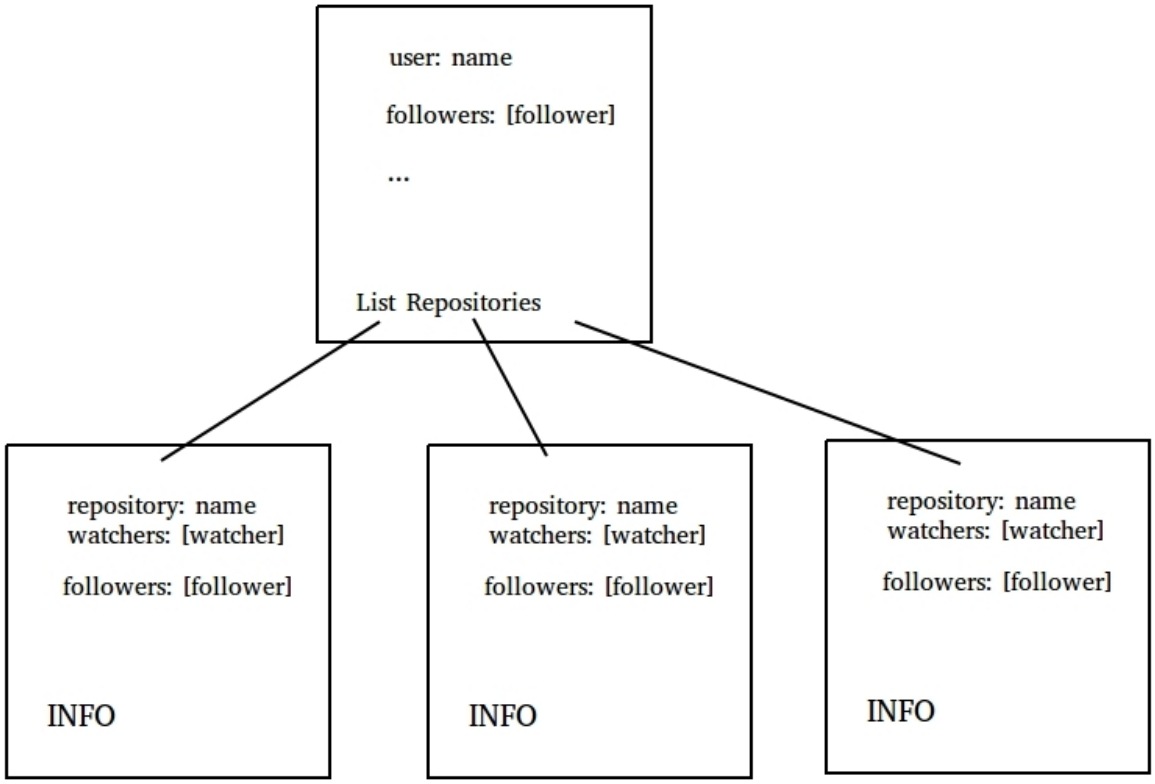


Рис. 2: Структура данных полученная по API

Получаем мы их через следующий запрос API github (для краткости некоторые поля удалены, а также убраны некоторые флаги запроса):

```
{
  user(login: "login whose information we want retrieve") {
    pinnedRepositories() {
      #считываем количества и имена pinned repositories
      #это те репозитории, которые пользователь хочет показать
    }
    name # имя пользователя
    login # логин пользователя
    location # страна пользователя
    followers {
    #считываем количество подписчиков
      totalCount
    }
    following{
    #количество пользователей в подписке
      totalCount
    }
    starredRepositories{
```

```
    #считываем количество репозиториев помечанных звездой
      totalCount
    }
    company
    organizations() {
      #считываем имена и количество организаций
    }
    contributedRepositories() {
    #получаем все репозитории пользователя
      totalCount #их количество
      edges { # итерируемся по всем репозиториям
        node {
          id
          name # имя репозитория
          watchers {
            totalCount #количество наблюдателей за ним
          }
          forks {
            totalCount # количество копий" репозитория"
          }
          stargazers {
            totalCount # количество звёзд, поставленных репозиторию
          }
          isFork #является ли это копией другого репозитория?
          description # краткое описание
        }
      }
    }
  }
}
```

В REST API нам пришлось бы сделать запросы по количеству сущностей, в то время как в GraphQL хватит одного. Количественное отношение потребления ресурса API может доходить 1 к 100 в пользу GraphQL. В ответ мы получим JSON, в котором будут все нужные для нас поля.

Таким образом формируется набор данных.

Набор данных всех пользователи Германии:

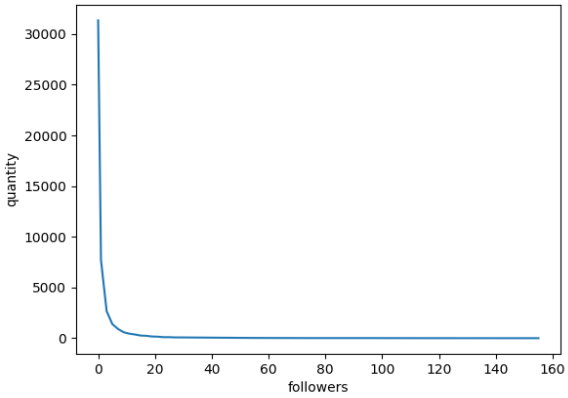
Все пользователи Германии были получены через поисковое API GitHub (как указано выше). Всего пользователей на момент написания было 58358, из них были выбраны только те, кто имеет как минимум 8 подписчиков, таких оказалось 7345. Данное ограничение было вызвано тем, что мы не хотели брать в рассмотрение людей, которые просто имеют аккаунт, а также пытались включить в dataset наиболее активных пользователей , тем самым желая ограничить влияние социальных эффектов для дальнейшего исследования.

4.2. Анализ пользователей

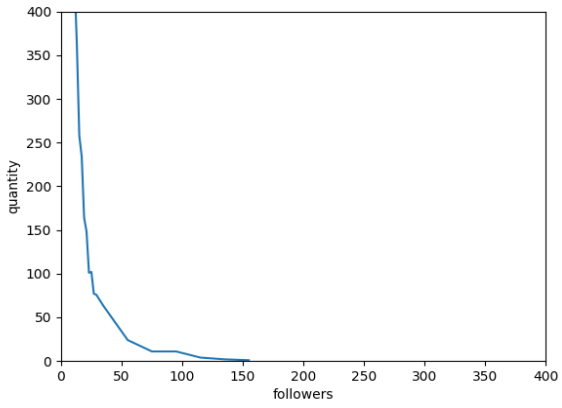
Ниже будет проведён анализ пользователей Германии. Будет показано соотношение между количеством пользователей и числом подписчиков у них, также будет показано соотношение между числом подписчиков и реальным вкладом программиста, в конце будет представлено соотношение между характеристиками, входящими в репозиторий: количество наблюдателей (watchers), количество людей, поставивших звезду (stargazers), количество веток разработки (forks). Данные наблюдения не только предоставят понимание о внутренних взаимосвязях реальных пользователей, но и дадут глубокое понимание осмысленности метрики, приведённой в предыдущей главе.

Количество подписчиков:
Количество подписчиков - величина, которая часто используется как наивный метод оценки программистов. Давайте покажем, как много программистов имеют фиксированное количество подписчиков.

Ниже представлены графики, которые могут дать понимание количественного отношения пользоватлей, к числу подписчиков у них (группировка по количеству подписчиков). Отношение представлено в разном маштабе:



(a) отношение количество пользователей к количеству подписчиков



(b) это же отношение на меньшем маштабе

Рис. 3: кривая распространения подписчиков (Oy - количесвто пользователей, Ox - количество подписчиков у пользователя)

Как видим, количество программистов резко убывает с ростом числа подписчиков, поэтому выбор подхода оценки пользователя по количеству подписчиков может показаться осмысленным. В следующем пункте будет продемонстрировано, как число подписчиков связано с реальной работой программиста.

Соотношение profile estimation и contribution estimation:

В этом разделе рассмотрим взаимосвязь ключевых характеристик пользователя по нашей метрике. Ниже представлен график соотношения оценки профиля (profile estimation) и оценки вкладов (contribution estimation). Напомним, что оценка профиля равна количеству подписчиков у пользователя, умноженных на числовой коэффициент, а оценка вкладов равна сумме всех взвешенных вкладов в какие-либо репозитории.

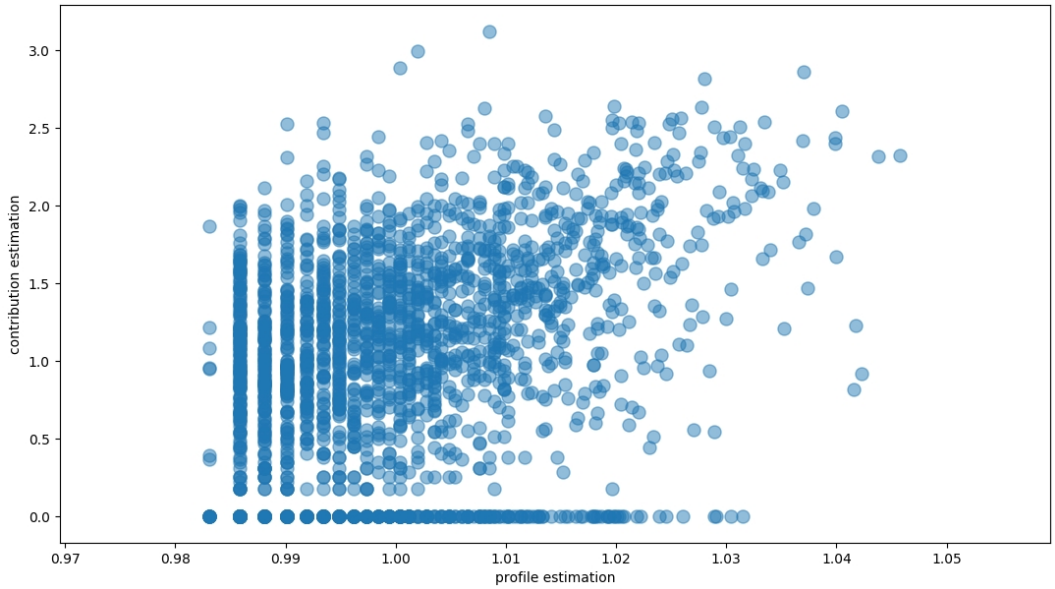


Рис. 4: соотношение составных частей метрики (точка графика - пользователь)

Как можно видеть из графика, в начале, когда количество подписчиков (followers) маленькое, соответственно маленькая и оценка профиля (так как $(profile\ estimation) = user.followers * ratio\_followers$) в свою очередь оценка вкладов может принимать практически любые значения, затем с повышением количества подписчиков, исключая явления, когда много подписчиков и совсем нет вкладов в другие репозитории, оценка профиля начинает немного расти вместе с оценкой всех вкладов (появляется слабая зависимость).

Это позволяет сделать вывод, что по оценке профиля нельзя судить о оценке вкладов (на графике видно, что величины независимы) и, следовательно, алгоритм, использующий для оценки только количество подписчиков, ненадёжен.

Соотношение между элементами contribution estimation:

Также необходимо исследовать соотношения между характеристиками, входящими в общую оценку вкладов (contribution estimation). Важно показать, как воздействуют данные характеристики друг на друга. Это имеет смысл не только для исследования общей картины взаимосвязи данных, но и для указания причины выбора именно

такого вида формулы.

$$Rr(r) = r.watchers * ratio\_watchers + (r.stargazers/1000) * r.forks$$

$$+r.stargazers * ratio\_stargazers$$

Перечислим их ещё раз: наблюдатели (watchers), звёзды (stargazers), копии (forks).

Также напомним, что сущностью, обладающей данными характеристиками, является репозиторий, поэтому точками на графике ниже будут представлены именно репозитории.

Ниже график, показывающий соотношение количества наблюдателей (watchers) и количества звёзд (stargazers).

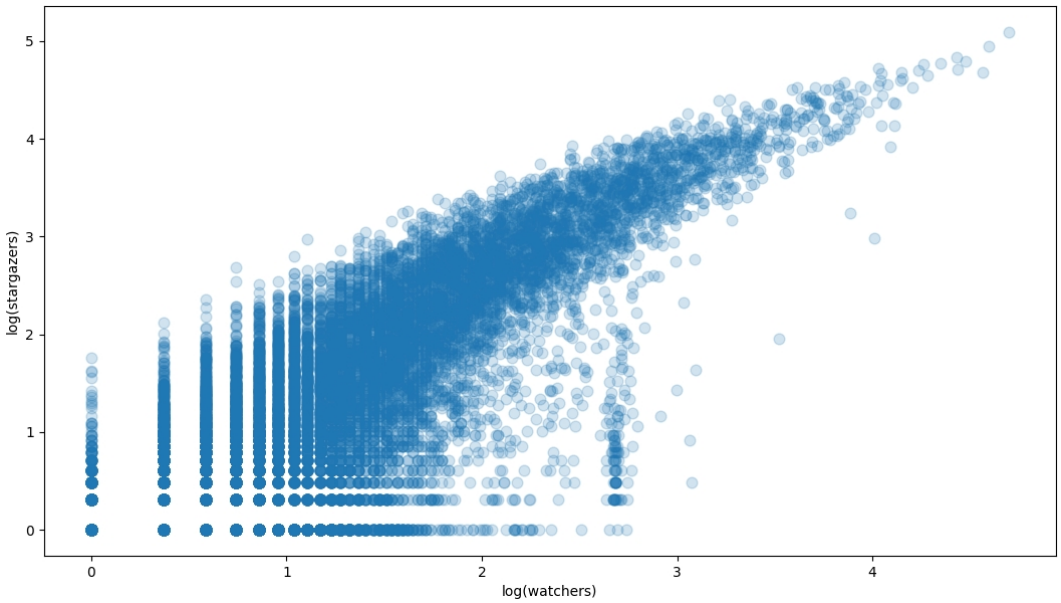


Рис. 5: логарифмированное отношение watchers и stargazers, точка на графике - репозиторий

Как мы можем видеть, при малых значениях характеристики имеют более свободные колебания относительно друг друга, но с увеличением они становятся прямо пропорциональны друг другу.

Теперь рассмотрим соотношение числа копий(forks) к количеству наблюдателей (watchers).

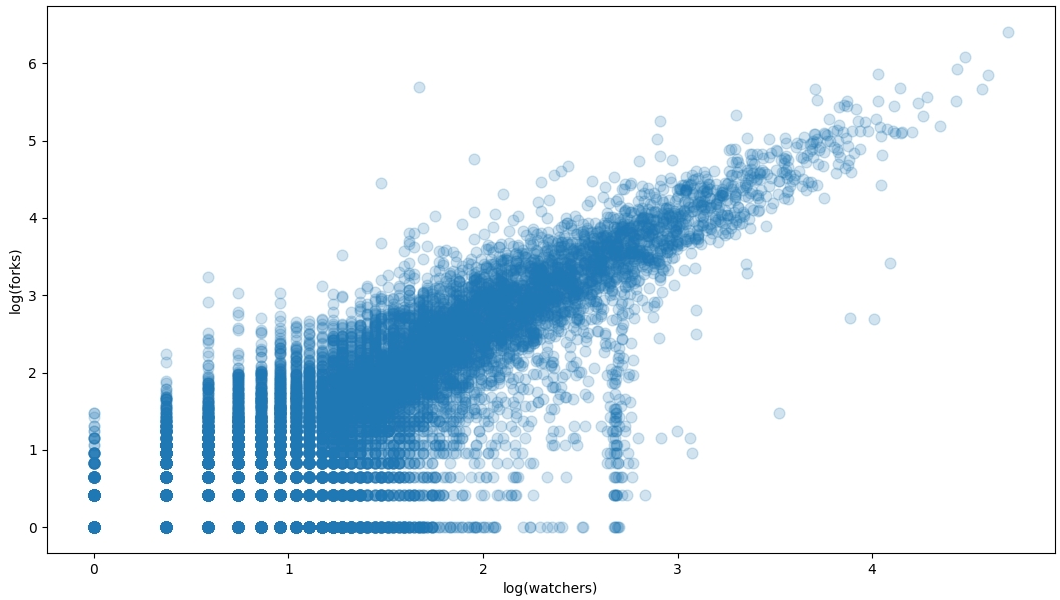


Рис. 6: логарифмированное отношение watchers и forks, точка на графике - репозиторий

Как можно видеть на рисунке выше, реализуется примерно такое же поведение, как между числом наблюдателей и числом звёзд, следовательно, число копий и число звёзд между собой имеют такое же соотношение, то есть все характеритики подчиняются одинаковому закону: при увеличении значения становятся прямо пропорциональны друг другу. Из этого следует вывод, что количество наблюдателей, количество копий и количество звёзд одинаково показывают популярность репозитория. Это также подкрепляет идею вычисления веса репозитория в виде линейного многочелена от данных элементов.

5. Проведение анализа самых популярных программистов мира

Набор данных ”самые популярные программисты мира” был собран с помощью GitHub API и имеет точно такое же внутреннее строение, как и элементы в наборе данных ”все пользователи Германии”. Критерием для оценки популярности было количество подписчиков (followers). На самых популярных пользователях был проведён анализ, который показал, что сумма взвешенных вкладов (contribution estimation) абсолютно не коррелирует с их популярностью (profile estimation), и делать какие-то выводы на данных пользователях не представляется возможным. Элементы характеризующие их сильно разбросаны, что объясняется тем, что получение такой высокой популярности могло быть связано с внешними факторами, как указано в работе [1]. Однако можно утверждать, что данные пользователи в большинстве своём имеют границу снизу по проделанной работе (contribution estimation), то есть мы можем с достаточно большой уверенностью гарантировать, что если человек будет пересекать какой-то порог по сумме взвешенных вкладов, то он тоже будет иметь высокую популярность. Тем самым удалось связать количество подписчиков и оценку снизу для суммы взвешенных вкладов. Заметим, что это очень интересное наблюдение, так как мы подкрепляем этим определение новой введенной нами метрики. Метрика показывает, что популярность на сайте Github на самом деле действительно непредсказуемая величина и может получать своё значение за пределами данного сервиса, но в то же время гарантирует, что если вы будете набирать значение по сумме взвешенных вкладов (contribution estimation) больше заданного порога, то вы также станете очень популярным на GitHub. Также заметим, что это взаимосвязь была не различима на обычных пользователях, как было указано в (4.2). Ниже рисунок, показывающий данный порог, все координаты были прологарифмированы для наглядности.

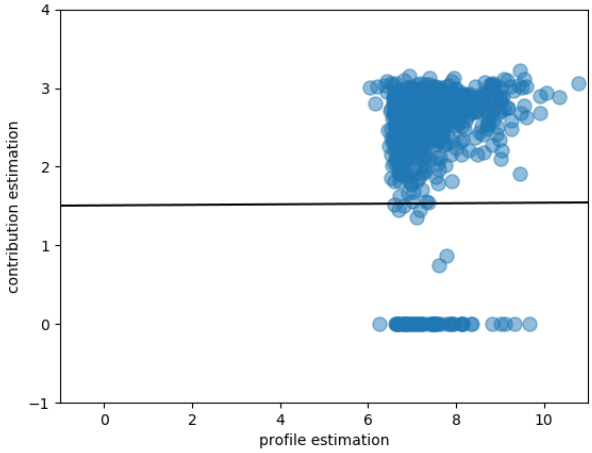


Рис. 7: самые популряные пользователи (contribution estimation и profile estimation)

6. Ручной алгоритм

Выше была построена метрика и показаны некоторые зависимости, объясняющие её состоятельность для оценки программиста (важную часть играет реальный вклад разработчика). Как и было указано ранее, теперь попытаемся решить вопрос нахождения ручного алгоритма для оценки программиста.

Можно подумать, для чего создавать ручной алгоритм, если у нас уже есть метрика для оценки пользователей? Если вы посмотрите в главу (3.1), вы поймёте, что крайне тяжело использовать метрику вручную. Однако, остаётся вопрос, а может ли человек, не прибегая к помощи компьютера, оценить программиста? Данная глава отвечает на этот вопрос, предлагая свой ручной алгоритм.

Заметим, что использование метрики при создании ручного алгоритма будет играть ключевое значение, так как построенный алгоритм ручного оценивания должен удовлетворять требованию осмысленности в контексте нашей предметной области, вследствие чего можно сделать вывод, что алгоритм ручного оценивания должен коррелировать оценку по метрики. Если бы метрика и ручной алгоритм оценивали программистов по-разному, то это говорило бы о несостоятельности одного из подходов.

Ниже будет продемонстрировано, какие характеристики, указанные во введение, со стартовой страницы профиля пользователя, влияют больше всего на оценку программиста, используя нашу метрику (6.1), далее исходя из выводов о самых значимых характеристиках будет продемонстрирован алгоритм ручного оценивания (6.2), использующий данные характеристики.

6.1. Самые важные характеристики со страницы профиля

Сначала напомним какие характеристики существуют на странице пользователя:

1)Organizations - организации в которых состоит пользователь;

2)Repositories - репозитории (репозиторий - ”папка с кодом” какого-либо проекта);

3)Stars - количество звёзд, которые поставил пользователь другим репозиториям. Также можно просмотреть непосредственно эти репозитории;

4)Followers - количество тех, кто подписался на данного пользователя. Также можно просмотреть непосредственно этих пользователей;

5)Following - количество людей в подписке пользователя с возможностью просмотреть профили этих людей;

6)Popular repositories - самые популярные репозитории пользователя, которые были специально выбраны для экспозиции их на главной странице. В настройках пользователь может поменять список выбранных репозиториев (тогда их называют pinned repositories, то есть специально поставленные пользователем);

7)Contribution - поле, характеризующие вклад пользователя в какие-либо проекты за последний год. Является натуральным числом.

.

Характеристики со страницы программиста не используются в формуле метрики, за исключением количества подписчиков (followers), но как мы уже видели в разделе (4.2 соотношение profile estimation и contribution estimation), сумма взвешанных вкладов и количество подписчиков практически независимые величины, поэтому количество подписчиков недостаточно, чтобы предсказать результат по метрики. Однако, используя остальные характеристики со страницы пользователя, предсказать оценку по метрики становится возможно. Необходимо ещё заметить, что характеристик со страницы профиля пользователя очень мало, следовательно, это даёт возможность создания практически полезного ручного алгоритма, так как вычислять в ручную оценку по метрики невозможно ввиду трудоёмкости.

Как были найдены данные характеристики:

С помощью метрики были размечены все пользователи Германии, затем была решена задача предсказания результата по метрики, используя в качестве входного вектора характеристики со страницы профиля пользователя. Структурой для предсказания был взят многослойный перцептрон, хотя это не критично для нашей задачи. После обучения на размеченных пользователях Германии $(user, ball)$, на выходе получили нейросеть, способную предсказывать результат метрики по характеристикам со страницы пользователя. Это позволило сделать вывод, что зависимость между метрикой и информацией со страницы пользователя есть (существует предсказатель, который может распознать эту зависимость).

Далее был выбран наименьший набор из этих характеристик, который также позволяет предсказать оценку пользователя по метрики, а значит и более всего влияет на эту оценку. Выбор наименьшего набора характеристик был сделан для удобства ручного алгоритма, который будет потом базироваться на этих характеристиках (меньше характеристик - проще ручной алгоритм).

В конце получили всего две характеристики: количество вкладов за год (поле Contribution) и количество подписчиков (поле Followers). Ниже представлен график, позволяющий понять точность предсказания результата метрики только по полям количество подписчиков и количество вкладов за год. По оси абсцисс отложены разработчики, пронумерованные числами, по оси ординат во сколько раз разнится оценка по метрики и предсказанная оценка по данным характеристикам.

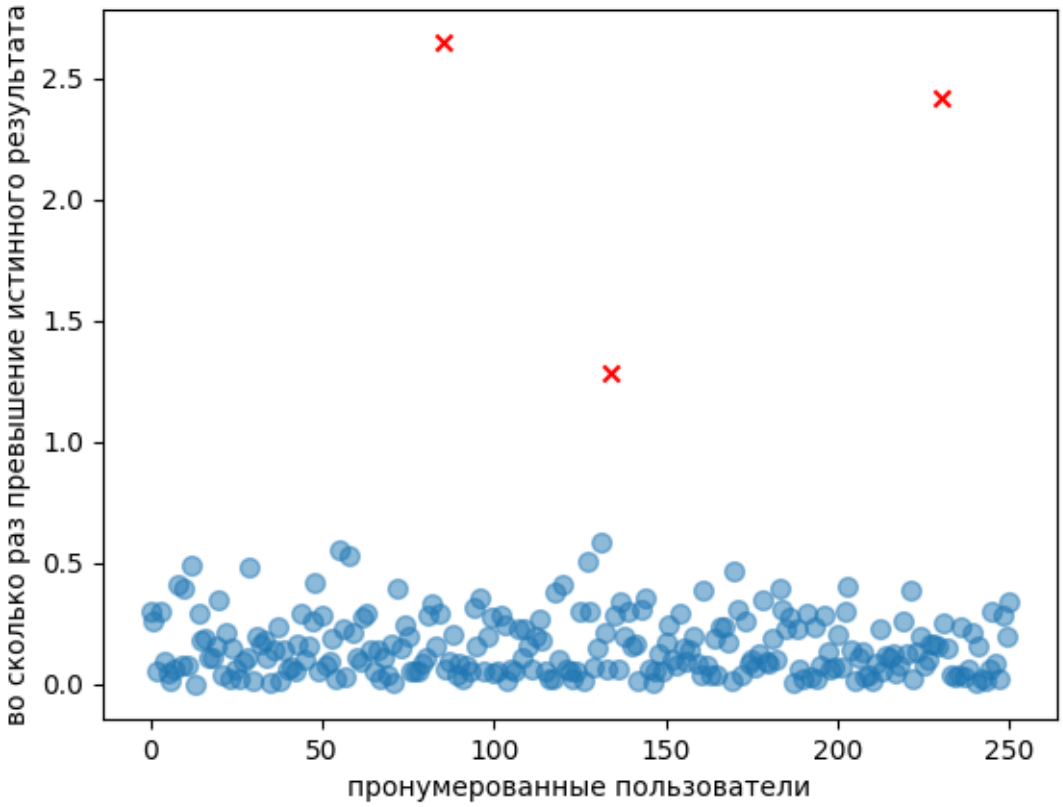


Рис. 8: разница пресказания по полям num_followers и contribution, от истинного результата метрики

Как видим оценка при предсказании отличается не более чем в 1.5 раза на большинстве пользователей (крестиками показаны в более чем два раза отличающиеся оценки). Как упоминалось выше можно делать вывод о связи количества подписчиков и количества вклада за год с реальной оценкой по метрике, следовательно, существует простой ручной алгоритм не более чем в 1.5 раза отличающегося предсказания оценки всего лишь по двум полям.

6.2. Ручной алгоритм

Ручной алгоритм оказался очень простым, ввиду доказательства соотношения между визуально доступными данными и оценкой по метрике, которая может быть подсчитана обходом всех репозиториев пользователя. В самом алгоритме необходимо использовать два поля: количество вкладов и количество подписчиков, для удобства также стоит прологарифмировать их. Функция оценки примет вид:

$$estimate(user) = F(log(len(user.followers)), log(user.contribution))$$

Где $log()$ - логарифм по натуральному основанию, $len()$ функция, возвращающая длину списка.

F может быть найдена решением задачи логистической регрессии на линейных функциях от данных характеристик. Было получено следующие соотношение на фиксированных параметрах метрики.

$$estimate(user) = 1.5 * log(len(user.followers)) + 1.4 * log(user.contribution)$$

Ниже представлен график, показывающий как разнится оценка по нашему ручному алгоритму от оценки по метрики (по оси абсцисс пронумерованные пользователи, по оси ординат во сколько раз разнятся оценки).

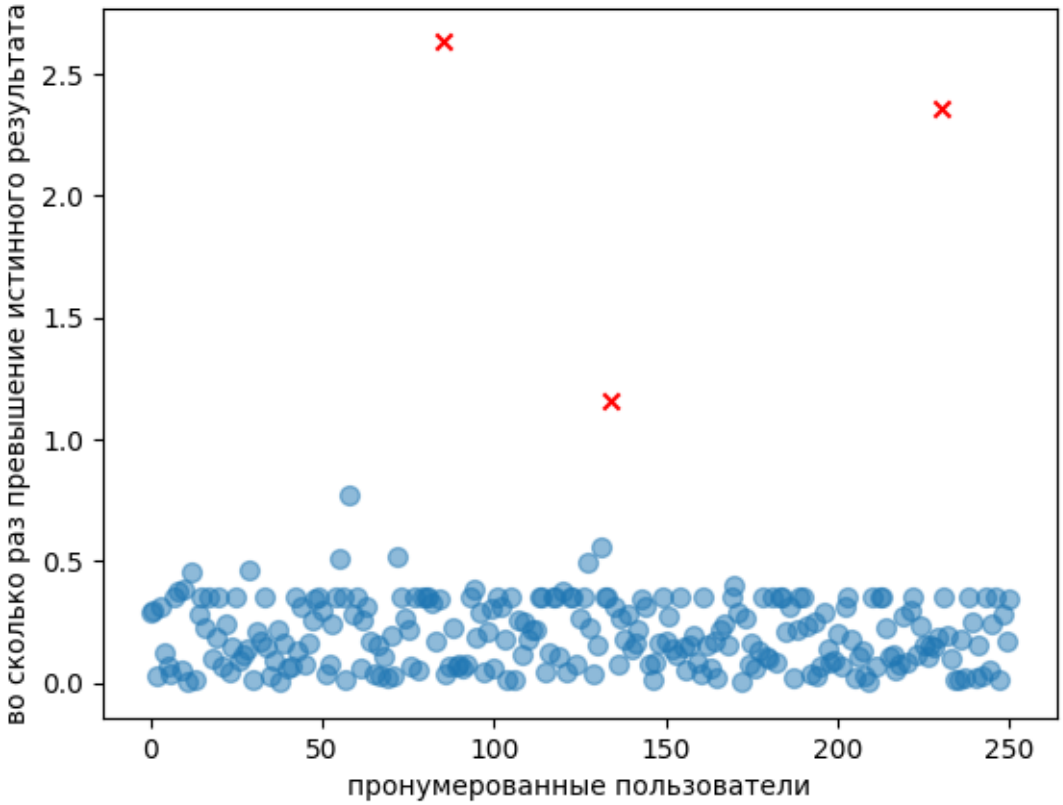


Рис. 9: разница пресказания по num_followers и contribution, от истинного результата метрики

Как видим данный ручной алгоритм довольно хорошо предсказывает оценку по метрики (отклоение более чем на 50% очень редко).

Вывод

Нам удалось построить ручной алгоритм, который может использовать HR менеджер, не прибегая к помощи компьютера. Построенный алгоритм намного проще,

чем продемонстрированная метрика и решает проблему невозможности применение данной метрики вручную. Заметим, что это очень важный результат, так как если принять оценку по метрики, как осмысленную в терминах GitHub, то ручной алгоритм должен приближать данную метрику, что и было продемонстрировано. Также, очевидно, что ручной алгоритм не должен быть слишком сложным, что также была учтено. В конце мы получили удобный для применения ручной алгоритм, а точнее даже правило о том, на какие параметры на стартовой страницы стоит смотреть в первую очередь при оценке разработчиков.

7. Заключение

В данной работе была продемонстрирована удобная с практической точки зрения метрика (3.1), которая позволяет оценивать программистов на сайте GitHub. Данная метрика делает акцент на реальных вкладах программистов, поэтому является корректной и осмысленной в нашей предметной области. Также в работе был проведён анализ немецких разработчиков (4), в том числе с использованием построенной метрики. В конце был приведён на практике реализуемый ручной алгоритм, который позволяет предсказать результат по метрики с довольно хорошей точностью (6.2), используя характеристики доступные на странице профиля.

Данная работа может служить руководством для внедрения подобной оценки в реальные системы поиска разработчиков, что сильно поможет в работе HR специалистам. Подходы продемонстрированные в этом тексте могут способствовать в доказательстве оправданности иных построенных метрик для сайта GitHub, а также в выработки ручных алгоритмов на основе построенных метрик.

Список литературы

[1] Exploring the Patterns of Social Behavior in GitHub / Yue Yu, Gang Yin, Huaimin Wang, Tao Wang // Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies. — CrowdSoft 2014. — 2014. — P. 31–36.

[2] Gousios Georgios, Kalliamvakou Eirini, Spinellis Diomidis. Measuring Developer Contribution from Software Repository Data // Proceedings of the 2008 International Working Conference on Mining Software Repositories. — MSR '08. — 2008. — P. 129–132.

[3] Hauff Claudia, Gousios Georgios. Matching GitHub Developer Profiles to Job Advertisements // Proceedings of the 12th Working Conference on Mining Software Repositories. — MSR '15. — 2015. — P. 362–366.

[4] Influence analysis of Github repositories / Yan Hu, Jun Zhang, Xiaomei Bai et al. // Published online. — 2016. — URL: `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4975729/`.

[5] Joicy Xavier, Autran Macedo, Marcelo de Almeida Maia. Understanding the popularity of reporters and assignees in the GitHub // Proceedings of the 26th International Conference on Software Engineering Knowledge Engineering, At Vancouver, Canada. — SEKE 2014. — 2014. — P. 484–489.

[6] Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository / Laura Dabbish, Colleen Stuart, Jason Tsay, Jim Herbsleb // Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. — CSCW '12. — 2012. — P. 1277–1286.

[7] Tsay Jason, Dabbish Laura, Herbsleb James. Influence of Social and Technical Factors for Evaluating Contribution in GitHub // Proceedings of the 36th International Conference on Software Engineering. — ICSE 2014. — 2014. — P. 356–366.